EDA, Data Clearning, and Feature Engineering

Importing modules and loading data

In [1]:

```
## EDA libraries
import pandas as pd;
import numpy as np;
import matplotlib.pyplot as plt
import seaborn as sns

# Text processing libraries
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
import string
from sklearn.feature_extraction.text import TfidfVectorizer

# Dimensionality reduction
import pca as pca

# Data imbalance
from imblearn.over_sampling import SMOTE
from collections import Counter

# Warnings
import warnings; warnings.filterwarnings('ignore')
```

In [2]:

```
train_text = pd.read_csv("training_text", sep="\|\|", engine="python", names=["ID","TEXT"], skiprows=1)
train_variants = pd.read_csv('training_variants')
```

Exploratory Data Analysis and Initial Data Cleaning

In [3]:

```
## Checking shape and head of trianing_text dataframe
display(train_text.head(3))
display(train_text.shape)
```

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...

```
(3321, 2)
```

In [4]:

```
## Checking shape and head of trianing_variant dataframe
display(train_variants.head(3))
display(train_variants.shape)
```

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2

```
(3321, 4)
```

We need to merge the two files on the given IDs.

Merging the text and variant file

In [5]:

```
## Merging text and variant information
train = pd.merge(train_text, train_variants).set_index('ID')
```

In [6]:

```
## Checking shape and head of merged dataframe
display(train.head(3))
display(train.shape)
```

ID	TEXT	Gene	Variation	Class
0	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A	Truncating Mutations	1
1	Abstract Background Non-small cell lung canc...	CBL	W802*	2
2	Abstract Background Non-small cell lung canc...	CBL	Q249E	2

```
(3321, 4)
```

Cheking NaN values in datapoints, if any

In [7]:

```
## Subseting the datapoints which have NAN values in some TEXTs
train[train.isna()['TEXT']]
```

Out[7]:

ID	TEXT	Gene	Variation	Class
1109	NaN	FANCA	S1088F	1
1277	NaN	ARID5B	Truncating Mutations	1
1407	NaN	FGFR3	K508M	6
1639	NaN	FLT1	Amplification	6
2755	NaN	BRAF	G596C	7

There are five datapoints with NAN values

Collecting TEXT for the datapoints with NAN values.

- These texts were collected from standard websites by a domain expert, the links are also mentioned as comments

In [8]:

```
text_1109 = "FANCA S1088F protein properly localizes to the nucleus, \
it alters FANC complex function, enhances sensitivity to DNA damaging agents, \
and sensitizes cells to PARP inhibitors in vitro and in vivo. his is consistent \
with previous reports that showed mutations in FANCA were associated with differing \
sensitivity to DNA cross-linking agents."
# https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5593159/


text_1277 = "the truncated ARID5B proteins lack these PEST sequences, but still have the ARID domain. \
Our study showed that the truncated ARID5B protein had longer half-life than that of wild-type ARID5B. \
Furthermore, we showed that the C-terminus of ARID5B had a repressive property, which was confirmed by a \
reporter gene assay with transient transfection. The C-terminus was deleted in the truncated ARID5B. \
The data suggest that the wild-type ARID5B could suppress the expression of down-stream target genes, \
but the truncated ARID5B could not. Taken together, the truncated ARID5B may accumulate in cells due to \
its longer half-life and inhibit repressive function of the wild-type ARID5B in a dominant-negative fashion. \
We also performed colony formation assays using an endometrial cancer cell line, Ishikawa. Expression of wild-type \
ARID5B was strongly suppressed in colonies resistant to G418, comparing with expression of GFP transfected into the \
cells as a control. In summary, we found that the truncated ARID5B is a long half-life protein without its \
transcriptional property, which may inhibit the transcriptional property of wild-type ARID5B"
# https://aacrjournals.org/cancerres/article/74/19_Supplement/2469/594378


text_1407 = "FGFR3 K508M lies within the protein kinase domain of the Fgfr3 protein. \
K508M confers a loss of function to the Fgfr3 protein as demonstrated by induction of growth arrest \
in cell culture and inactivation of Stat1 in vitro and loss of kinase activity in the context of Fgfr3-Tacc3."
# https://ckb.jax.org/geneVariant/show?geneVariantId=10473


text_1639 = "13q12.3, Receptor tyrosine kinase/growth factor signaling, Amplification, \
FLT1 Amplification is present in 0.39% of AACR GENIE cases, with colon adenocarcinoma, \
rectal adenocarcinoma, colorectal adenocarcinoma, breast invasive ductal carcinoma, and \
invasive breast carcinoma having the greatest prevalence"
# https://www.mycancergenome.org/content/alteration/flt1-amplification/

text_2755 = "7q34, Kinase fusions, MAP kinase signaling, Substitution Missense,  Exon 15, BRAF, Protein kinase, Deleterious\
BRAF G596C is present in 0.02% of AACR GENIE cases, with lung adenocarcinoma, breast invasive ductal carcinoma,\
colorectal mucinous adenocarcinoma, melanoma, and rectal adenocarcinoma having the greatest prevalence"
# https://www.mycancergenome.org/content/alteration/braf-g596c/#ref-4
```

In [9]:

```
## Adding the collected text to the resepective locs
train.iloc[1109, 0] = text_1109
train.iloc[1277, 0] = text_1277
train.iloc[1407, 0] = text_1407
train.iloc[1639, 0] = text_1639
train.iloc[2755, 0] = text_2755
```

In [10]:

```
train
```

Out[10]:

	TEXT	Gene	Variation	Class
ID				
0	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A	Truncating Mutations	1
1	Abstract Background Non-small cell lung canc...	CBL	W802*	2
2	Abstract Background Non-small cell lung canc...	CBL	Q249E	2
3	Recent evidence has demonstrated that acquired...	CBL	N454D	3
4	Oncogenic mutations in the monomeric Casitas B...	CBL	L399V	4
...	...	...	...	...
3316	Introduction Myelodysplastic syndromes (MDS) ...	RUNX1	D171N	4
3317	Introduction Myelodysplastic syndromes (MDS) ...	RUNX1	A122*	1
3318	The Runt-related transcription factor 1 gene (...	RUNX1	Fusions	1
3319	The RUNX1/AML1 gene is the most frequent targe...	RUNX1	R80C	4
3320	The most frequent mutations associated with le...	RUNX1	K83E	4

3321 rows × 4 columns

```
In [11]:
```

```
# Shape of the data after adding missing text and before dropping NANs
display(train.shape)

# Dropping NANs
train = train.dropna()

# Shape of the data after dropping NANs
train.shape
```

```
(3321, 4)
```

```
Out[11]:
```

```
(3321, 4)
```

```
In [12]:
```

```
train
```

```
Out[12]:
```

ID	TEXT	Gene	Variation	Class
0	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A	Truncating Mutations	1
1	Abstract Background Non-small cell lung canc...	CBL	W802*	2
2	Abstract Background Non-small cell lung canc...	CBL	Q249E	2
3	Recent evidence has demonstrated that acquired...	CBL	N454D	3
4	Oncogenic mutations in the monomeric Casitas B...	CBL	L399V	4
...	...	...	...	...
3316	Introduction Myelodysplastic syndromes (MDS) ...	RUNX1	D171N	4
3317	Introduction Myelodysplastic syndromes (MDS) ...	RUNX1	A122*	1
3318	The Runt-related transcription factor 1 gene (...	RUNX1	Fusions	1
3319	The RUNX1/AML1 gene is the most frequent targe...	RUNX1	R80C	4
3320	The most frequent mutations associated with le...	RUNX1	K83E	4

3321 rows × 4 columns

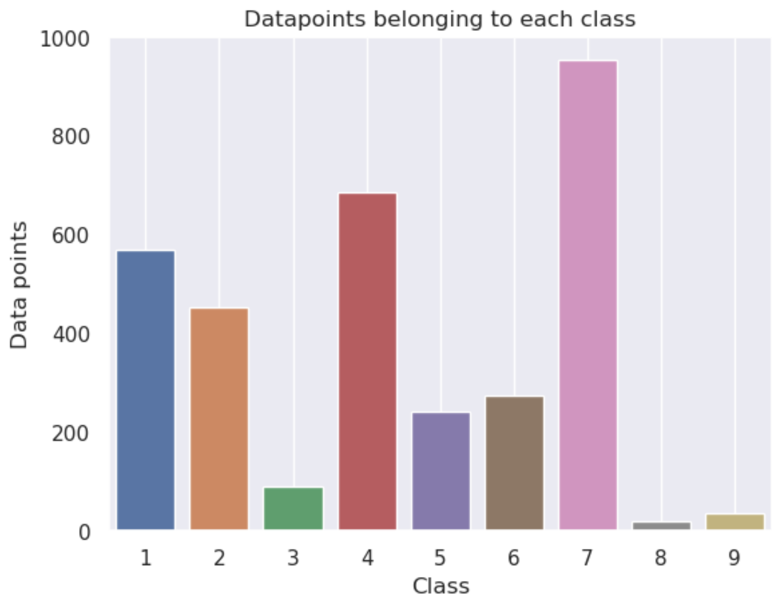
No NANs left in the dataset

Visualization

Bar plot for class-counts

In [13]:

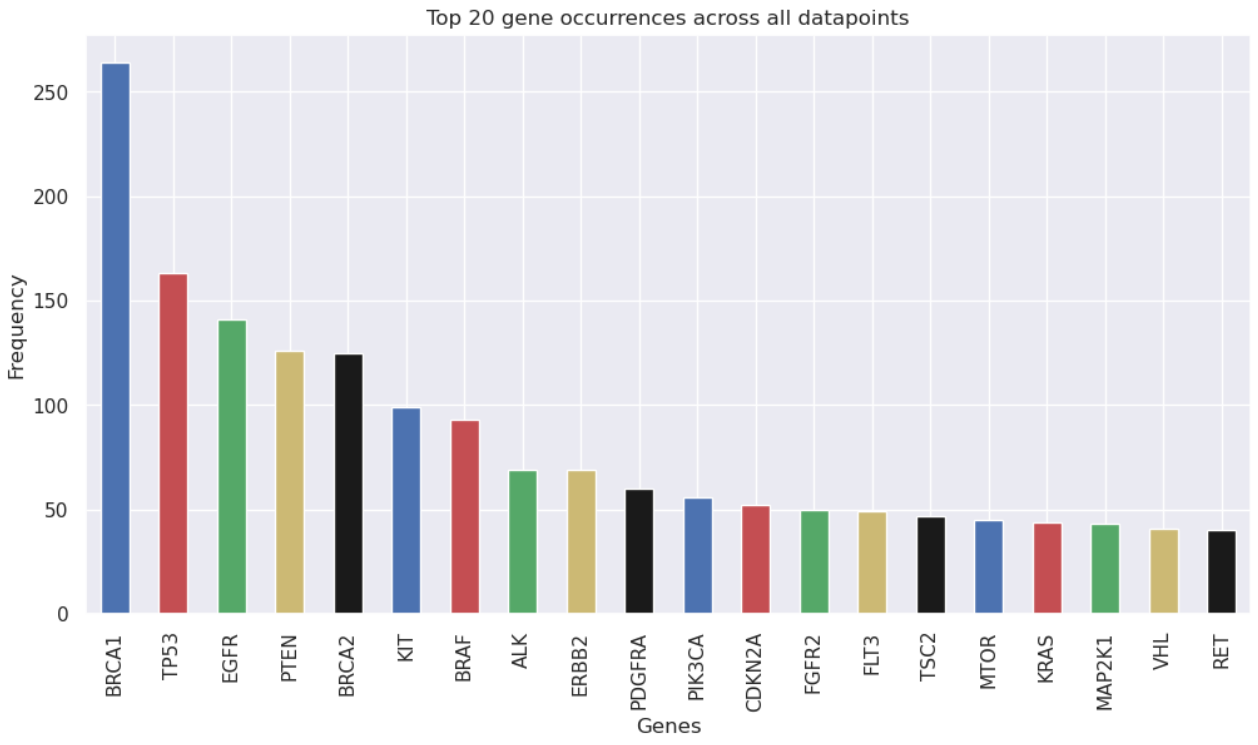
```
#Count Plot of classes(0-9)
class_distribution = train['Class'].value_counts().sort_index()
class_distribution = class_distribution.reset_index().T.drop(index = 'index')
class_distribution.columns = range(1,10)
sns.set_theme(font_scale = 1)
sns.barplot(class_distribution)
plt.xlabel('Class')
plt.ylabel('Data points')
plt.title("Datapoints belonging to each class")
plt.grid()
plt.show()
```



Top 20 gene occurrences

In [14]:

```
    # Gene counts for intial 20 data points
    my_colors = ['b', 'r', 'g', 'y', 'k']
    plt.figure(figsize=(12,6))
    df_gene_plot = train['Gene'].value_counts()[:20].plot(kind='bar',
                        color = my_colors, y ='Class',stacked = True)
    plt.title("Top 20 gene occurrences across all datapoints")
    plt.xlabel('Genes'); plt.ylabel('Frequency')
    plt.show()
```

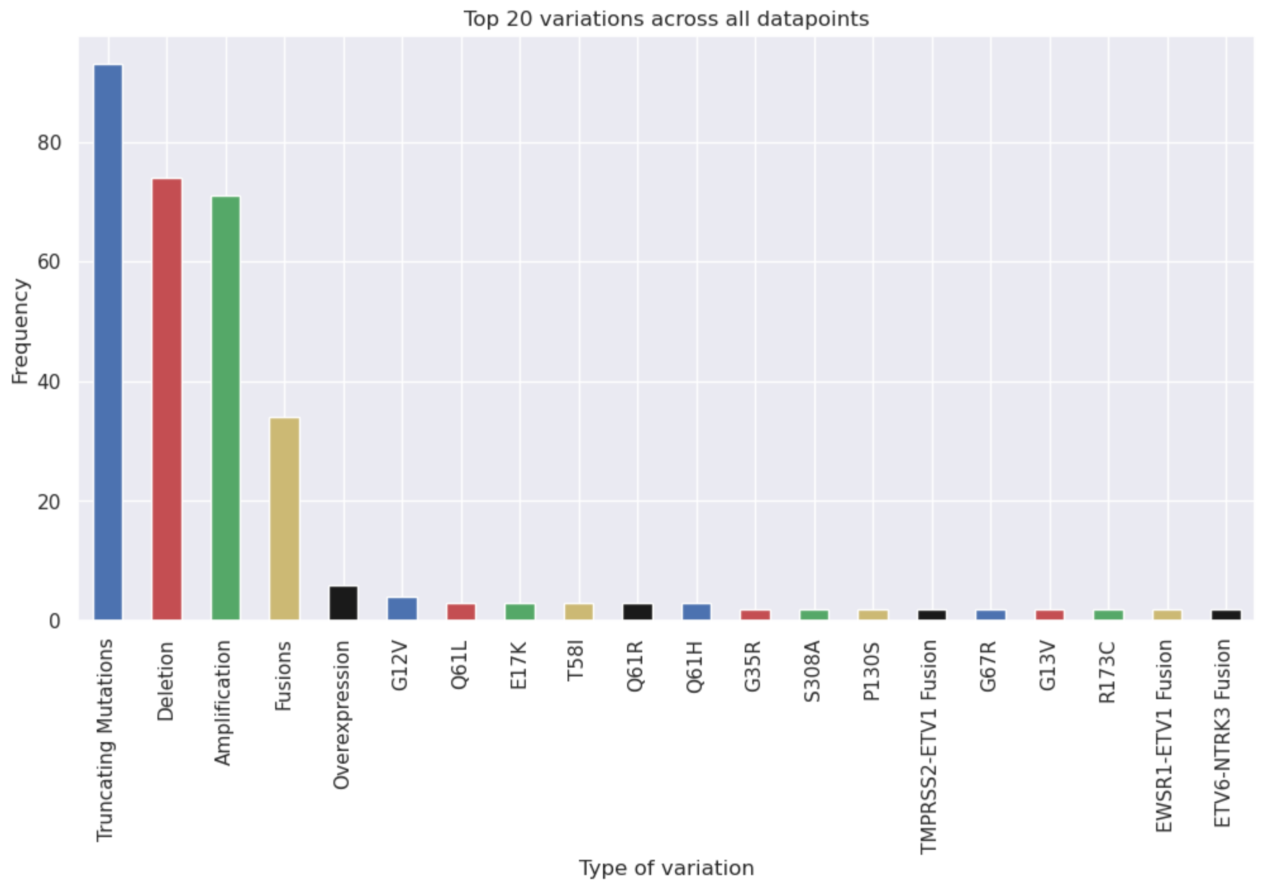


Top 20 variations

In [15]:

```
#Top 20 Frequent Variations
plt.figure(figsize=(12,6))

df_gene_plot = train['Variation'].value_counts()[:20].plot(kind='bar'
             ,color = my_colors, y ='ID', ylabel = 'Frequency')
plt.title("Top 20 variations across all datapoints")
plt.xlabel('Type of variation'); plt.ylabel('Frequency')
plt.show()
```



In [ ]:

```

```

Scatterplot

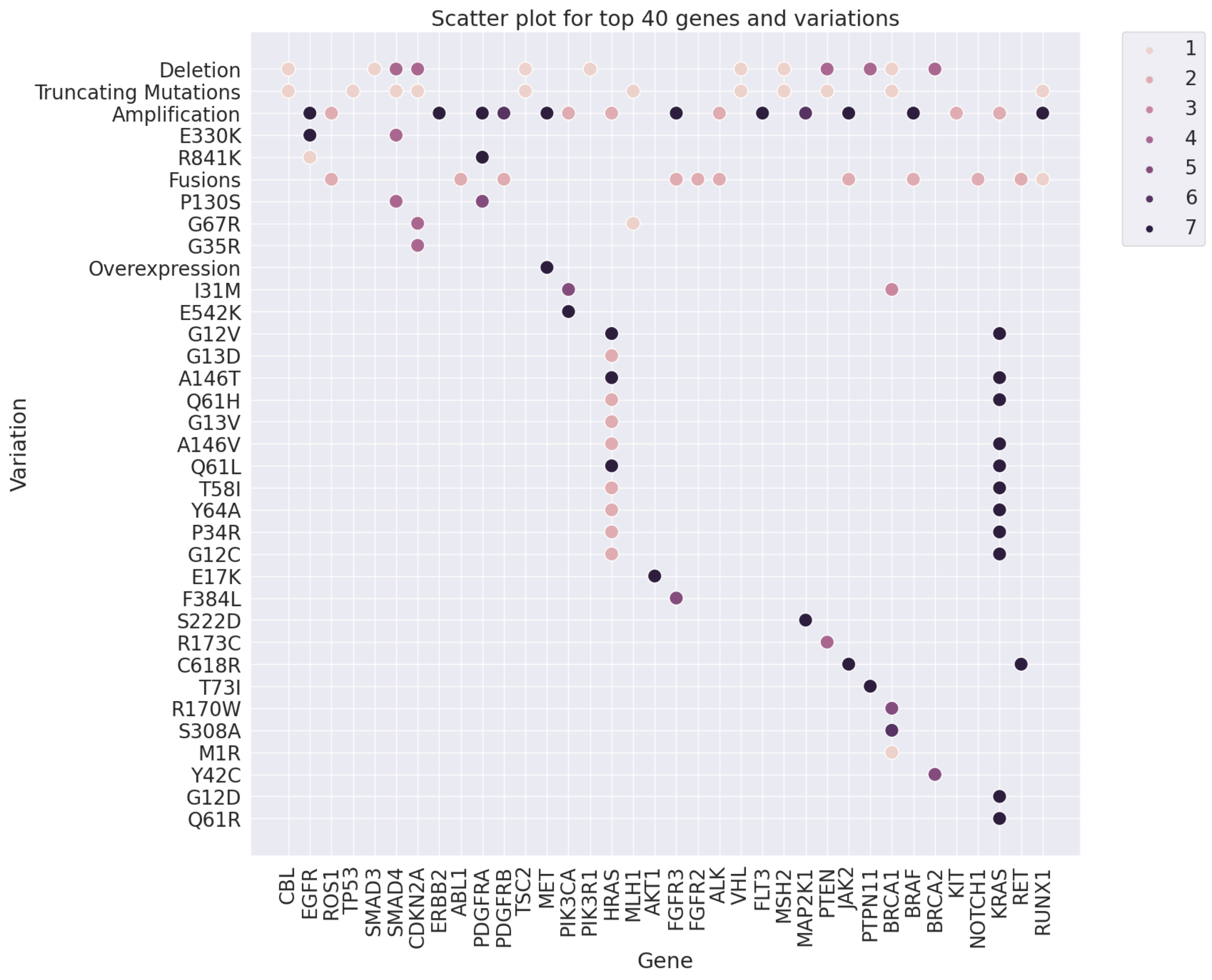
In [16]:

```
top_variations = train['Variation'].value_counts()[:40].index
top_genes = train['Gene'].value_counts()[:40].index
top_gene_var_df = train[train['Variation'].isin(top_variations)][train['Gene'].isin(top_genes)]
```

In [17]:

```
sns.set_theme(font_scale = 1.8) # set theme for plots

df = top_gene_var_df
plt.figure(figsize=(15, 15))
sns.scatterplot(data=df,
                x="Gene", y="Variation", hue="Class", s = 200,)
plt.xticks(rotation = 90)
plt.title("Scatter plot for top 40 genes and variations")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)
plt.show()
```



Feature Engineering

1. Extracting text based features

In [18]:

```
stop_words = set(stopwords.words('english'))
global stop_words
```

Text preprocessing

In [19]:

```
def nlp_preprocessing(sentence):

    # empty string to contain the final one
    final_string = ""

    # lowering the sentence
    sentence = sentence.lower()

    # removing the punctuation
    no_punc_sentence = "".join([i for i in sentence if i not in string.punctuation])

    # making tokens
    tokens = word_tokenize(no_punc_sentence)

    # filtering the tokens of stop words
    filtered_tokens = [w for w in tokens if not w in stop_words]

    # stemming
    ps = PorterStemmer()
    final_tokens = [ps.stem(i) for i in filtered_tokens]


    try:
        #converting all tokens to string
        for index in final_tokens:
            final_string += index
            final_string += " "
        return final_string
    except:
        print("Couldn't convert into string, hence returing tokens")
        return final_tokens
```

In [20]:

```
# collecting all strings in a list
list_of_strings = [nlp_preprocessing(i) for i in train['TEXT'].values]
```

Vectorizing the text by TF-IDF

In [21]:

```
# defining TF-IDF
tfidf = TfidfVectorizer(min_df = 2, ngram_range=(1, 2), max_features = 700)
```

In [22]:

```
tfidf_result = tfidf.fit_transform(list_of_strings)
```

In [23]:

```
train_df_tfidf = pd.DataFrame(tfidf_result.toarray(),index=train.index, columns = tfidf.get_feature_names_out())
train_df_tfidf.head()
```

Out[23]:

	05	10	100	11	12	13	14	15	16	17	...	wild	wild type	wildtyp	within	witho
ID																
0	0.004691	0.047508	0.007141	0.012937	0.003147	0.010144	0.006562	0.021852	0.013463	0.003447	...	0.005115	0.005149	0.006451	0.009700	0.0111
1	0.022286	0.059951	0.029682	0.015365	0.018690	0.012048	0.007793	0.014830	0.011993	0.004094	...	0.000000	0.000000	0.026816	0.003840	0.0132
2	0.022286	0.059951	0.029682	0.015365	0.018690	0.012048	0.007793	0.014830	0.011993	0.004094	...	0.000000	0.000000	0.026816	0.003840	0.0132
3	0.006805	0.077531	0.005179	0.014075	0.027393	0.024525	0.019037	0.018114	0.009765	0.005000	...	0.000000	0.000000	0.074863	0.009381	0.0323
4	0.000000	0.039565	0.015858	0.007183	0.010485	0.026283	0.003643	0.013866	0.007475	0.003828	...	0.011359	0.011435	0.064470	0.014361	0.0082

5 rows × 700 columns

Dimensionality reduction using PCA

In [24]:

```
pc = pca.pca(n_components = 0.95, normalize = False)
```

In [25]:

```
train_pca = pc.fit_transform(train_df_tfidf)
```

```
[pca] >Processing dataframe..
[pca] >The PCA reduction is performed to capture [95.0%] explained variance using the [700] columns of the input data.
[pca] >Fit using PCA.
[pca] >Compute loadings and PCs.
[pca] >Compute explained variance.
[pca] >Number of components is [191] that covers the [95.00%] explained variance.
[pca] >The PCA reduction is performed on the [700] columns of the input dataframe.
[pca] >Fit using PCA.
[pca] >Compute loadings and PCs.
[pca] >Outlier detection using Hotelling T2 test with alpha=[0.05] and n_components=[191]
[pca] >Outlier detection using SPE/DmodX with n_std=[2]
```

In [26]:

```
train_pca
```

Out[26]:

```
{'loadings':              05        10       100        11        12        13        14  \
 PC1    -0.006448 -0.013576 -0.005835 -0.001285 -0.001018  0.000671  0.003649
 PC2     0.003538  0.002099  0.001466 -0.014345 -0.009121 -0.011350 -0.011131
 PC3     0.001906 -0.025889 -0.003813 -0.022487 -0.018011 -0.017272 -0.015534
 PC4    -0.011564 -0.036174 -0.009666 -0.001508 -0.011300 -0.009265 -0.010826
 PC5     0.003019 -0.002086 -0.003150 -0.007588 -0.006296 -0.004934 -0.003511
 ...          ...       ...       ...       ...       ...       ...       ...
 PC187 -0.011777  0.055172 -0.048714  0.027705  0.043017  0.039209  0.055761
 PC188  0.018543  0.037156  0.020309 -0.001028 -0.029070 -0.007628 -0.068941
 PC189 -0.008647 -0.062110 -0.003938  0.004356 -0.014174 -0.015972 -0.039922
 PC190 -0.003805  0.042099  0.005266  0.001433  0.022810  0.021551  0.046695
 PC191  0.001639 -0.024230 -0.015202 -0.013218 -0.001458  0.018977  0.004481

              15        16        17  ...      wild  wild type   wildtyp  \
 PC1    -0.000272  0.003766  0.002617  ... -0.000837  -0.000663  0.009615
 PC2    -0.001027 -0.004750 -0.012561  ...  0.009386   0.010104  0.030599
 PC3    -0.012360 -0.013478 -0.014228  ...  0.001585   0.002418  0.016728
 PC4    -0.016676 -0.010711 -0.007711  ... -0.021564  -0.021628 -0.028135
```

In [27]:

```
topfeat_df = pd.DataFrame(pc.results['topfeat'])
reqfeat_df = topfeat_df.head(pc.n_components)
reqfeat_df.head()
```

Out[27]:

	PC	feature	loading	type
0	PC1	brca1	0.582415	best
1	PC2	et al	0.361428	best
2	PC3	pten	0.673727	best
3	PC4	mutat	-0.311229	best
4	PC5	p53	0.568809	best

In [28]:

```
print('Following are the number of top features which remained after PCA: ')
display(len(reqfeat_df['feature'].unique()))
```

```
Following are the number of top features which remained after PCA:

116
```

In [29]:

```
top_feat_list = reqfeat_df['feature'].unique().tolist()
top_feat_list
```

Out[29]:

```
['brca1',
 'et al',
 'pten',
 'mutat',
 'p53',
 'egfr',
 'imatinib',
 'alk',
 'fusion',
 'smad4',
 'tsc2',
 'flt3',
 'mtor',
 'ra',
 'nrf2',
 'fgfr2',
 'pdgfra',
 'erbb2',
```

In [30]:

```
train_df_reduced = train_df_tfidf[top_feat_list]
train_df_reduced.head()
```

Out[30]:

	brca1	et al	pten	mutat	p53	egfr	imatinib	alk	fusion	smad4	...	model	erlotinib	analysi	transloc	nm	fold	
ID																		
0	0.0	0.000000	0.0	0.069545	0.00000	0.000000	0.0	0.0	0.026292	0.0	...	0.010370	0.0	0.049963	0.0	0.005701	0.000000	0.0
1	0.0	0.004765	0.0	0.402661	0.04328	0.334401	0.0	0.0	0.000000	0.0	...	0.008211	0.0	0.045378	0.0	0.000000	0.000000	0.0
2	0.0	0.004765	0.0	0.402661	0.04328	0.334401	0.0	0.0	0.000000	0.0	...	0.008211	0.0	0.045378	0.0	0.000000	0.000000	0.0
3	0.0	0.000000	0.0	0.462382	0.00000	0.009498	0.0	0.0	0.022884	0.0	...	0.010029	0.0	0.123638	0.0	0.000000	0.000000	0.0
4	0.0	0.000000	0.0	0.543787	0.00000	0.159960	0.0	0.0	0.000000	0.0	...	0.084444	0.0	0.019581	0.0	0.000000	0.017184	0.0

5 rows × 116 columns

2. Extracting gene and variation based features

One Hot Encoding

In [31]:

```
ohe_gene_var = pd.get_dummies(train.drop(columns = ['TEXT']), columns=['Gene', 'Variation'], drop_first=True)
```

In [32]:

```
ohe_gene_var.head()
```

Out[32]:

	Class	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	...	Variation_Y87N	Variation_
ID													
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	2	0	0	0	0	0	0	0	0	0	...	0	
2	2	0	0	0	0	0	0	0	0	0	...	0	
3	3	0	0	0	0	0	0	0	0	0	...	0	
4	4	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 3259 columns

3. Stacking the text and gene-variation based features

In [33]:

```python
stack = pd.merge(ohe_gene_var.reset_index(), train_df_reduced.reset_index()).set_index('ID')
stack
```

Out[33]:

	Class	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	...	model	erlotinib	analysi	transl
ID															
0	1	0	0	0	0	0	0	0	0	0	...	0.010370	0.0	0.049963	0.0000
1	2	0	0	0	0	0	0	0	0	0	...	0.008211	0.0	0.045378	0.0000
2	2	0	0	0	0	0	0	0	0	0	...	0.008211	0.0	0.045378	0.0000
3	3	0	0	0	0	0	0	0	0	0	...	0.010029	0.0	0.123638	0.0000
4	4	0	0	0	0	0	0	0	0	0	...	0.084444	0.0	0.019581	0.0000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3316	4	0	0	0	0	0	0	0	0	0	...	0.065400	0.0	0.095324	0.0554
3317	1	0	0	0	0	0	0	0	0	0	...	0.102372	0.0	0.078752	0.0157
3318	1	0	0	0	0	0	0	0	0	0	...	0.000000	0.0	0.033125	0.4917

Removing data imbalance

Applying smote to tackle imbalance

In [34]:

```python
stack.head()
```

Out[34]:

	Class	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	...	model	erlotinib	ana
ID														
0	1	0	0	0	0	0	0	0	0	0	...	0.010370	0.0	0.049
1	2	0	0	0	0	0	0	0	0	0	...	0.008211	0.0	0.045
2	2	0	0	0	0	0	0	0	0	0	...	0.008211	0.0	0.045
3	3	0	0	0	0	0	0	0	0	0	...	0.010029	0.0	0.123
4	4	0	0	0	0	0	0	0	0	0	...	0.084444	0.0	0.019

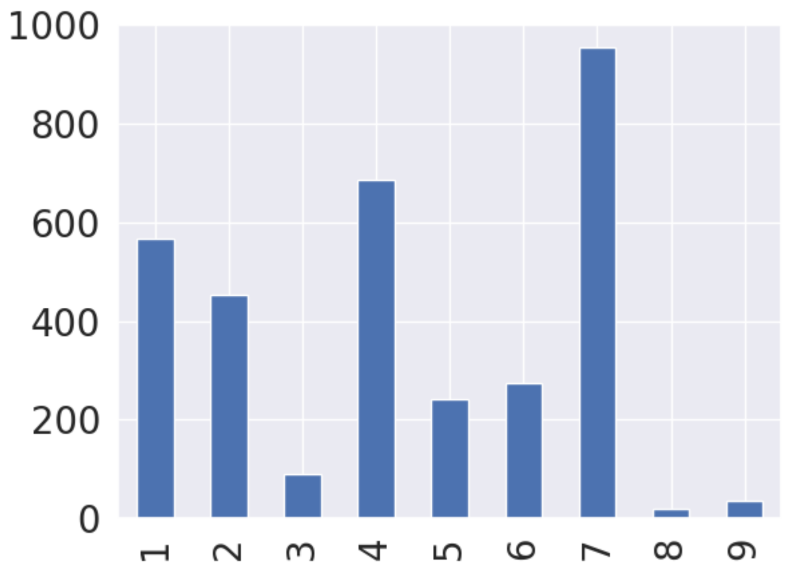
5 rows × 3375 columns

In [35]:

```python
X_stack = stack.copy()
y_stack = X_stack.pop('Class')
```

In [36]:

```
stack['Class'].value_counts(sort=False).plot(kind = 'bar')
plt.show()
```



In [37]:

```
counter = Counter(y_stack)
counter
```

Out[37]:

```
Counter({1: 568, 2: 452, 3: 89, 4: 686, 5: 242, 6: 275, 7: 953, 8: 19, 9: 37})
```

In [38]:

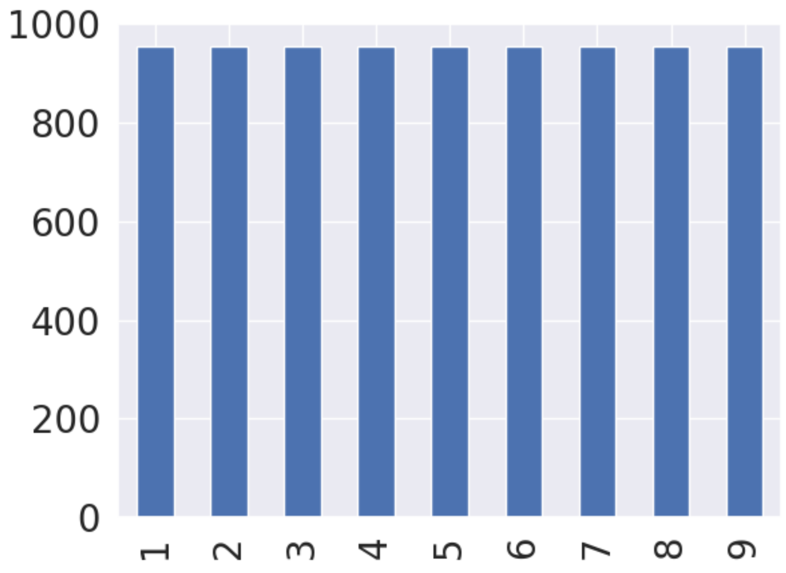
```
oversample = SMOTE()
```

In [39]:

```
X, y = oversample.fit_resample(X_stack, y_stack)
```

In [44]:

```
y.value_counts(sort=False).plot(kind = 'bar')
plt.show()
```



In [45]:

```
display(X.head())
display(X.shape)
```

	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	Gene_ARID1A	...	model	erlotinib
0	0	0	0	0	0	0	0	0	0	0	...	0.010370	0.0
1	0	0	0	0	0	0	0	0	0	0	...	0.008211	0.0
2	0	0	0	0	0	0	0	0	0	0	...	0.008211	0.0
3	0	0	0	0	0	0	0	0	0	0	...	0.010029	0.0
4	0	0	0	0	0	0	0	0	0	0	...	0.084444	0.0

5 rows × 3374 columns

```
(8577, 3374)
```

X

In [46]:

```
X.insert(0 ,'Class', y)
```

In [ ]:

```
X # this is the df we will use to do training testing
```

In [47]:

```
X.shape
```

Out[47]:

```
(8577, 3375)
```

In [48]:

```
X
```

Out[48]:

	Class	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	...	model	erlotinib	a
0	1	0	0	0	0	0	0	0	0	0	...	0.010370	0.0	0.0
1	2	0	0	0	0	0	0	0	0	0	...	0.008211	0.0	0.0
2	2	0	0	0	0	0	0	0	0	0	...	0.008211	0.0	0.0
3	3	0	0	0	0	0	0	0	0	0	...	0.010029	0.0	0.1
4	4	0	0	0	0	0	0	0	0	0	...	0.084444	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8572	9	0	0	0	0	0	0	0	0	0	...	0.054776	0.0	0.0
8573	9	0	0	0	0	0	0	0	0	0	...	0.021582	0.0	0.0
8574	9	0	0	0	0	0	0	0	0	0	...	0.031599	0.0	0.0
8575	9	0	0	0	0	0	0	0	0	0	...	0.004533	0.0	0.0
8576	9	0	0	0	0	0	0	0	0	0	...	0.022249	0.0	0.0

8577 rows × 3375 columns

In [ ]:

```
# X.to_csv('final_trianing_frame2.csv', index = False)
```

In [ ]:

```

```